Energy-Efficient IaaS-PaaS Co-design for Flexible Cloud Deployment of Scientific Applications

David Guyon, Anne-Cécile Orgerie, Christine Morin

Energy-Efficient IaaS-PaaS Co-design for Flexible Cloud Deployment of Scientific Applications

David Guyon, Anne-Cécile Orgerie and Christine Morin
Univ Rennes, Inria, CNRS, IRISA, Rennes, France
Email: david.guyon@irisa.fr, anne-cecile.orgerie@irisa.fr, christine.morin@inria.fr

***Abstract*—Reducing the massive amount of energy consumed by cloud datacenters becomes of major importance. In the usual approach where resources are consolidated into fewer servers in order to power down the others, it still remains periods of time when servers are not fully utilized. Consequently, it exists unused resources that are not exploited although they could be used to execute applications compatible with the variable availability of these resources. In this work, we propose a cloud system where the Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) layers interact to find execution trade-offs that exploit the unused resources at IaaS level. PaaS users are involved in the energy optimization by proposing to delay their executions and adapt resource sizes in order to fit with the available unused resources. Our evaluation by simulation is based on real data and expresses a realistic large scale cloud scenario. Results show that according to the proportion of energy-aware users, this system is able to reduce the amount of servers by using resources that would have been wasted otherwise. Therefore, our solution allows datacenters to consume less energy than with usual resource managers where all applications start their execution at submission time with their initial resource size.**

***Keywords*-Cloud computing; Infrastructure-as-a-Service; Platform-as-a-Service; energy**

I. Introduction

Estimations show that by 2030 the energy consumed by datacenters will represent 13% of the worldwide energy consumption [1]. Due to climate change and the increasing success of Internet services, priorities should be given to the reduction of their energy consumption.

In cloud computing, studies have been conducted to increase datacenters energy efficiency [2]. Their efforts focus on optimizing energy consumption at Infrastructure-as-a-Service (IaaS) level, as it is closer to hardware resources than the Platform-as-a-Service (PaaS) layer, and thus closer to the electricity consumption itself. However, several studies already showed that higher energy savings are possible when enhancing interactions between these two cloud layers [3], [4]. IaaS knows about the availability of hardware resources and can deliver energy-related information that could help the PaaS layer to make energy-aware decisions. In return the PaaS layer could inform IaaS providers on users' applications flexibility in order to help the consolidation process. Typically, when PaaS users ask for virtual resources, requested resources are made available as soon as possible. However, some users could accept their request to be handled differently if it saves energy by either delaying the deployment or changing the size of the requested resources (and consequently the duration). While this approach is not compatible with applications that continuously execute (web jobs), time-bound scientific applications could exhibit flexibility on starting time and resource size as long as results arrive before a deadline and if total cost does not increase. This scenario is realistic as scientists running HPC applications are looking at clouds as a cost effective alternative to HPC [5].

Our objective is to reduce IaaS datacenter energy consumption with a cooperation allowing PaaS to express the flexibility of its applications and IaaS to inform on when and how many resources are predicted to be unused. This way, energy savings are achievable by shifting and resizing some applications on these otherwise unused resources.

Our proposition is a PaaS-IaaS co-design that offers to PaaS users energy-efficient execution trade-offs. This co-design allows users to decide the flexibility level they agree to give to their execution. Our evaluation by simulation, based on real datacenter traces and real application execution logs, shows that when all users deploy applications that are flexible in time and size, it is easier for IaaS providers to allocate resources to applications without the need to power up additional servers. Up to 5.49% of energy is saved compared to a scenario already consolidating the workload, powering down unused servers and where all users prefer their application execution to start at submission time.

The rest of the article is organized as follows. Section II presents the context and list the assumptions. The system architecture is described in Section III and evaluated by simulation in Section IV. Discussion on limitations is given in Section V. Section VI discusses the related work. Finally, we conclude in Section VII.

II. Context and Assumptions

In this study, we consider a PaaS cloud with users deploying scientific applications. These applications are CPU-intensive, execute in a single Virtual Machine (VM) and have a finite execution time. Applications with a continuous execution (i.e. web application) are not considered here as

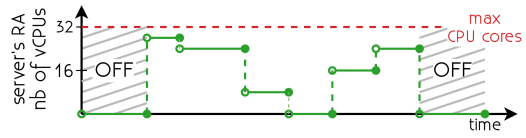

Figure 1: Example of a server's *Resource Availability* fluctuating according to the arrival and departure of resource reservations.

they cannot be delayed or changed in size without impacting users' satisfaction.

The PaaS negotiates with multiple IaaS providers to provide virtual resources to its users. We assume that PaaS users can choose to be tolerant to delaying their application execution. This flexibility in time has a limit defined in the PaaS Service Level Agreement (SLA) as the *time window* (TW). Its value defines the maximum number of hours the execution of an application can be delayed. As application executions can be delayed, we assume that IaaS resources can be reserved in advance (within TW).

VMs have a fixed size in terms of vCPUs, RAM and disk usage (called *flavor*), that remains unchanged during their lifetime (no dynamic resource scaling). However, here we simplify the problem by defining *flavors* only by their number of vCPUs, the most important component for CPU-intensive applications.

VMs energy consumption can be estimated in advance and computed more precisely after their execution. For this calculation to be possible, we make the assumption that IaaS providers know the power profile of their servers. IaaS providers optimize reservation placements with an allocation algorithm that targets the use of a minimum number of servers. In addition, each provider tries to reduce its energy consumption by shutting down unused servers [2] and putting them back online when required.

Based on utilization traces and because workloads tend to repeat themselves over time (usually on a daily basis), we assume that IaaS providers can predict the workload of their servers [6]. The load is defined as the amount of reserved vCPUs per server. Additionally, the power state of each server is also known. In other words, it is possible to record when servers are online or not (powered up/down) during the day, thus providing information on when resources are active or not. With this, it is possible to compute the per-server *Resource Availability* (RA). This value represents the amount of unbooked vCPUs when servers are powered up. As shown in Figure 1, RA fluctuates from 0 to the vCPU capacity of the server according to the arrival and departure of resource reservations. When RA reaches 0, it is either because the server is powered down or because all vCPUs are allocated. When all vCPUs are available, it means the server is not used. In this case, the server is powered down, thus making resources unavailable (RA falls back to 0 at the end of Figure 1).

Additionally, we make the following assumptions:

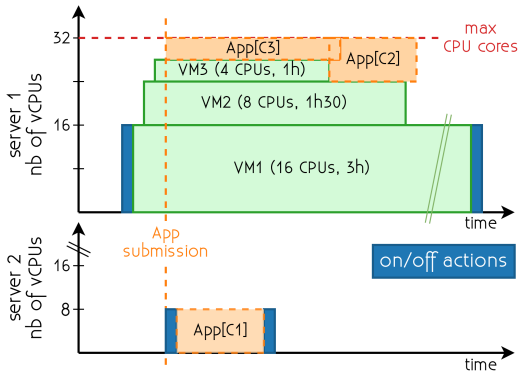

Figure 2: Possible scheduling (*execution contracts*) of an 8 vCPUs application in an infrastructure with 2 servers and 3 VMs. Starting the application at submission time (C1) requires to turn on server 2. Delaying (C2) or changing the size (C3) can avoid the need of server 2, thus saving energy.

- users know the execution time of their applications for each VM flavor. Typically, scientific applications are CPU-intensive and executed multiple times, so users can conveniently know their execution duration depending on the vCPUs number and the data input size. Moreover, predicting application's execution time for varying number of vCPUs in the VM configuration has been shown to be possible [7] ;
- each IaaS provider has its own incoming workload in addition to the one induced by the studied PaaS ;
- virtual resources are given for a specific duration and are released at the end of this period ;
- requested virtual resources are considered to be fully booked, regardless of whether the application really use them or not. CPU overcommitment is not considered.

III. Proposition

A. General Idea

In this work, we propose to benefit from the exchange of information between PaaS and IaaS layers, as well as the users' willingness to change resource size and to delay their application execution in order to achieve energy savings.

The general idea to achieve energy savings is presented in Figure 2. Based on the RA information, IaaS providers are able to propose to their users *execution contracts* (EC) with a delayed execution in order to fit with the availability of resources. An EC corresponds to a spatial and temporal placement of a VM on servers that stays within the maximum TW hours tolerated delay. Orange rectangles in Figure 2 depict three different ECs consuming different amounts of energy. Two servers owned by an IaaS provider host a total of three VMs. Server 1 is able to host all VMs, which explains why server 2 remains powered down. A user desires to execute her 8 vCPUs application for a duration of 30 minutes. When the VM request occurs, the only way to execute the application directly is to power up server

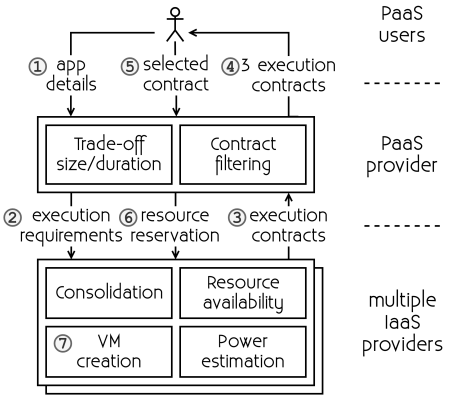

Figure 3: Detailed system architecture with the components of both cloud layers and the interactions between them and the end-user.

2 ($App[C1]$). This is the most energy consuming contract. Delaying the execution of the application of about 1 hour avoids to use server 2 ($App[C2]$) and reduces the energy consumption. It is also possible to execute the application at submission time without powering up the second server by changing the size to 4 vCPUs ($App[C3]$). In this simple use-case, we consider that attributing half of the requested vCPUs doubles the execution time of the application. This third contract has the same energy cost as the second contract but execution results are available sooner.

B. System Architecture Overview

In this section, we present the architecture of our system. Details on each component are provided in the following sections. The complete system architecture is presented in Figure 3. Each step of the overall process given in the following list is linked to the labels attached to the figure:

1) a user sends a request to the PaaS provider to execute her application ;
2) the PaaS provider sends several requests adjusted to the user flexibility to multiple IaaS providers ;
3) each IaaS provider proposes an EC for each request it received ;
4) the PaaS provider filters the ECs based on their energy cost and delay in order to propose three contracts to the user ;
5) the user selects the contract she wants to execute her application with ;
6) the PaaS provider informs the corresponding IaaS provider that its contract has been selected ;
7) this IaaS provider plans the execution of the application in a VM as defined in the EC.

PaaS users send the following details: the application, the size S in number of vCPUs (within the available flavors listed in Table I), and a table of execution lengths according to each possible size ($D[S]$).

In a previous work [8], we already showed that proposing to IaaS users to variate their resource size helps to reduce datacenters energy consumption. Here, the PaaS provider acts like an IaaS user and sends 4 different *execution requirement* (ER) requests to its collection of known IaaS providers:

- asks for a VM with S vCPUs ($flavor^{base}$) for a duration of $D[flavor^{base}.cpu]$ seconds without delay (as soon as the application submission arrives) ;
- asks for the same size and duration but allows a maximum delay of TW hours ;
- asks for the flavor one step larger than $flavor^{base}$ for a duration of $D[flavor^{base+1}.cpu]$ seconds within a maximum of TW hours delay ;
- same as previous requirement but with a flavor one step smaller than $flavor^{base}$ and a duration of $D[flavor^{base-1}.cpu]$ seconds.

This mechanism, handled by the *trade-off size/duration* component of our architecture, is based on the duration table given by the user.

Whenever an IaaS provider receives an ER request, its *consolidation* component searches in its collection of servers for the most energy efficient EC. If no delay is tolerated, it searches for a server with just enough available vCPUs at the current time. This is a bin packing problem. We use a greedy algorithm that gives priority to servers that are already active. The scheduling is optimized in order to have the highest amount of reserved vCPUs on the server. It avoids small VMs to "pollute" servers with large free spaces that could be used by larger VMs. If there are not enough available vCPUs on active servers, the system powers up a new one.

If delay is tolerated, the system uses the *resource availability* component that allows to know the availability of unused vCPUs in the next TW hours. It provides the delay value of a possible time slot for a VM of the requested size that stays within the tolerated deadline. If no time slot can be found, the process informs the PaaS layer and falls back to a scheduling where no delay is tolerated. More details on the *resource availability* are given in Section III-C.

Each EC comes with an estimation of the VM energy cost. This estimation is based on the power profile of servers, the VM size and its duration. Details on this calculation are given in Section III-D.

The *consolidation* component may find several ECs for a single ER request. Instead of sending them all to the PaaS layer, each IaaS provider executes a first filtering round to only send the most relevant. Contracts are projected in a graph plotting the energy prediction for the various delay. Data normalization is used in order to have axes with equivalent weight. We select the contract with the shortest Euclidean distance with the (0, 0) coordinates because this is the one with an equally balanced trade-off between delay and estimated energy cost.

To summarize, IaaS providers include in their EC the starting time t_{start} that is between $t_{submission}$ and

$t_{submission} + TW$ and the energy E^{VM} that the VM is estimated to consume.

For each ER request sent by the PaaS, IaaS provider that has been contacted submits its proposition of EC. At this stage, the *contract filtering* component selects the most energy-efficient EC received for a given ER. It searches for the contract that is the most equally balanced between delay and energy consumption. This filtering feature is the same as the one used for the first filtering round executed at the IaaS layer (trade-off between delay and energy).

Finally, end-users are given a choice between three ECs where they can choose the one they prefer:

- C1: application execution starts at $t_{submission}$ without changing the requested amount of resources ;
- C2: application execution is delayed of up to TW hours and the requested resources are not changed ;
- C3: application execution is delayed of up to TW hours and the given resources are one size larger or smaller than originally requested.

Each EC displays its starting and completion time ($t_{start} + D$). Delayed contracts (C2 and C3) display the percentage of energy they save in comparison with the contract starting directly (C1). A percentage of energy saved is provided instead of real values given by IaaS providers because revealing real values could break the separation of concerns between cloud layers. In the case of C3, it is also required to show the new resource configuration.

Each EC is valid for acceptance during a certain amount of time (few minutes) defined in the IaaS SLA as the *user's response time*. The PaaS user has to select the contract that suits the most her need within this time. All contracts that are not accepted within the allowed response time are released. Her choice is made in terms of execution time and depending on her motivation in reducing her impact on energy consumption. When an EC is accepted, the PaaS layer informs the corresponding IaaS provider to schedule the VM creation. If the selected contract is still valid, the IaaS provider executes two actions: the VM creation is scheduled at time t_{start} as specified in the EC and resources taken by the VM are removed from the RA schedule (resources pre-allocation). Updating RA is required because the resources that have just been pre-allocated will not be available when the VM will be created. In the case where the selected contract is not valid anymore, the PaaS has to restart its negotiation.

At time t_{start}, the *VM creation* component creates the VM on the server given by the *consolidation* component.

Next, we detail each component of our architecture.

C. Resource Availability Management

The *resource availability* informs on when the vCPUs of a server are expected to be available over time based on its historical workload. The computation of RA is based on the IaaS incoming workload excluding the PaaS workload.

Each IaaS provider generates a RA prediction for each server it owns. A per-server RA differs from a RA at the datacenter's level because it allows to know the largest VM a server can host. As an example, with an aggregated RA at infrastructure level, when 10 vCPUs are predicted to be available, it is impossible to know if it is 5 servers with 2 available vCPUs each or a single server with 10 unused vCPUs.

Each time a VM is instantiated or removed from a server, the new amount of vCPUs that are available on this server is recorded. We define n_s as the total number of vCPUs on a server s and the function $usage_s(t)$ that returns its number of vCPUs reserved at time t. The RA function can then be defined as: $RA(t) = n_s - usage_s(t)$.

The *consolidation* algorithm goes through the RA of each server in order to find a time slot for a requested VM. A valid time slot corresponds to a period of time in a server's RA when at least S vCPUs are available for a duration of minimum $D[S]$ seconds. At first, the algorithm searches on each server for time slots with an availability duration that is the closest to the application duration. Then, following a greedy algorithm, in this sub-list of possible time slots, only the one with the smallest difference between the amount of available vCPUs and the requested size S is retained. This time slot, along with its energy estimation cost (described in Section III-D) are included in the EC returned to the PaaS.

An IaaS provider has to update the corresponding server's RA when one of its proposed contract is selected, because of resources pre-allocation. The amount of vCPUs at the time slot specified in the contract is subtracted from the vCPUs that were predicted to be available. The RA schedule evolves over time as contracts are accepted.

D. Execution Contract Energy Estimation

In order to let the PaaS layer know which contract is the less energy consuming, each EC includes an estimation of the VM energy cost.

Our VM energy estimation is the sum of two parts. The first one is a portion of the idle energy consumption of the server. This energy is shared with the VMs currently hosted on the server according to their size. The second energy value is the maximum dynamic energy that the VM can consume. The dynamic energy is based on the application execution duration, the requested number of vCPUs and the power consumption of a single vCPU. We consider this CPU-based power model because the processor is known to be the most energy consuming component of servers [9] and this work targets CPU-intensive applications. If a server needs to be powered up for a VM, this energy cost is not included in the VM energy calculation.

This energy estimation requires two power values for a given server s: P^s_{idle} and P^s_{CPU}. P^s_{idle} is the power consumed by a server while not being used (i.e. not hosting any VMs) and P^s_{CPU} is the power consumption of 1 vCPU used at

its maximum capacity. Power consumption values can be measured with wattmeters attached to the servers. P^s_{idle} is reached each time there are no VMs running on the server. The measurement of P^s_{CPU} relies on P^s_{max} the maximum consumption with all vCPUs. The calculation of the CPU power is defined as follows:

$$P^s_{CPU} = (P^s_{max} - P^s_{idle})/n_s \quad (1)$$

Following, the equation to calculate the idle energy consumption of a VM.

$$E^{VM}_{idle} = P^s_{idle} \times \int_{t_{start}}^{t_{start}+D[S]} \frac{S}{n_s - RA(t) + S} \, \mathrm{d}t \quad (2)$$

The ratio refers to the requested number of vCPUs (S) over the total number of allocated and requested vCPUs. It reaches its closest value to 0 when the smallest VM size runs in a fully used server, and equals to 1 when any VM size is running alone on the server. The integral gives the average ratio over the VM lifetime.

The maximum dynamic energy the VM is expected to consume is calculated with: $E^{VM}_{dyn} = P^s_{CPU} \times S \times D[S]$.

Finally, the total amount of joules a VM is estimated to consume corresponds to the sum of idle and dynamic energies: $E^{VM} = E^{VM}_{idle} + E^{VM}_{dyn}$

The *power estimation* component allows to have an estimation of the VM energy consumption. However, this estimation can differ from the energy consumed after execution. As a matter of fact, the ratio used when calculating E^{VM}_{idle} evolves according to the update of the $RA(t)$ function. Therefore, it causes the energy cost to be sometimes overestimated.

IV. Evaluation

A. Experimental Setup

To validate our approach a simulator has been developed. The simulated PaaS application is the Montage scientific application (detailed in Section IV-A2) that is executed on real VMs of all possible sizes in order to get the different execution time lengths. Our system considers the flavors listed in Table I. It is based on the types of instances that can be found on the Amazon EC2 platform. Finally, workloads used at both IaaS and PaaS levels are traces from real cloud systems (detailed in Section IV-A3). They define the starting time, the size and duration of each VM.

Table I: List of VM flavors considered in the simulator with their Amazon EC2 hourly pricing for the EU Paris region.

Flavor	vCPU (S)	EC2 cost
t2.small	1	$0.0264/h
c5.large	2	$0.101/h
c5.xlarge	4	$0.202/h
c5.2xlarge	8	$0.404/h
c5.4xlarge	16	$0.808/h

1) Server Model: Simulated servers are based on real servers that we use to run our scientific application. These servers, provided by the Grid'5000 platform [10], are Dell PowerEdge R430 with 16 cores, 64GB of memory and 600GB of disk and equipped with fine-grained wattmeters. They have an average P_{idle} of 75.83W and consume 8.81W when powered down. The originally measured P_{max} was 174.04W. However, IaaS servers are usually of a larger size in order to host an important amount of VMs. To account for this in the simulator, we double the number of CPU cores per server to 32. P_{max} is adapted accordingly to 272.25W. Based on Eq. 1, P_{CPU} is 6.14W. Finally, we also measured energy of on/off state transitions that are respectively 4.92Wh (143 sec) to power up and 0.11Wh (6 sec) to power down. In our evaluation, this server model is used by all IaaS providers.

The number of servers per IaaS, shown in Table II, is defined in order to handle the peak usage of resources defined by their workload.

2) Scientific Application: To load our PaaS cloud with realistic scientific applications, we use Montage [11] which is CPU-intensive. Montage is a toolkit for assembling images into custom mosaics specialized for astronomical images. By giving a space location and a high/width (in angular degrees), it downloads all the images from its servers that match with the given space area. Then, it detects similarities between all images and finally generates an image of the space area.

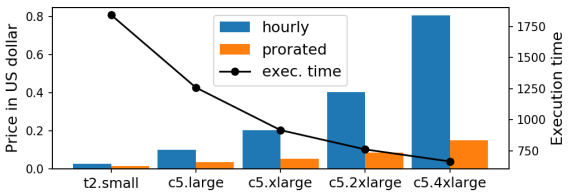

Figure 4: Execution time, hourly price and prorated price of Montage in all possible VM sizes.

In a preliminary experiment, we execute Montage in VMs of all possible sizes hosted on a real server. We build a specific Montage VM image optimized to be CPU intensive. This image contains the required input images and has a Montage installation using MPI in order to benefit from all available cores. This is a common PaaS behavior to have several VM images, each of them specialized for specific executions [12]. Figure 4 presents the execution time ($D[S]$) of Montage for each VM size. It also shows the price it would cost to execute it on Amazon EC2 instances in an hourly basis (pay for a full hour, even if reservation is less than an hour) and prorated (only pay for the exact reservation duration). This calculation corresponds to the flavor price given in Table I times the application execution time.

3) Real Datacenter Workloads: We use a real job submission trace as the input workload of each cloud provider. The Parallel Workloads Archive [13] provides open source

workload logs. The longest utilization log available (2 years long) in the archive is from the MetaCentrum Czech National Grid [14]. In order to find a typical working day (24h long workload) as an input of each cloud provider, a k-mean algorithm is executed to group the days with a similar job submission distribution. An analysis with 6 groups (k-mean with $k = 6$) exhibits one group with a typical working day trend (load during working hours). Four different days are taken from this group to represent the 4 IaaS workloads used in experiments (listed in Table II). All jobs in workloads are normalized in order to have a size within the available flavor sizes. Also, only the jobs with a duration between 5 min and 12 hours are retained. The duration of each IaaS job is rounded to hours because IaaS clouds, such as Amazon EC2, provide VMs on an hourly basis.

Table II: Details of the cloud layers.

Cloud layer	Nb of servers	Workload size
IaaS 1	60	2858 VMs
IaaS 2	60	2273 VMs
IaaS 3	250	3641 VMs
IaaS 4	40	1429 VMs
PaaS	N/A	1500 applications

For the PaaS layer, another workload is taken from the same k-mean group. This workload contains only 2, 4 and 8 vCPU jobs for easing the simulator task. Indeed, the C3 contract tries to scale up and down the resource size, thus we keep the larger and lower VM sizes for this contract.

B. Experimental Validation of our Approach

We validate our approach by simulation through 4 different experiments. The energy saving is in percent in comparison with the scenario where all PaaS applications execute directly without any modification (C1).

1) Energy Saving According to the Contract Selection: In this first experiment, we show the impact of the contract selection on the IaaS energy consumption and on the total monetary cost of applications. In this context, there is only one IaaS provider (*IaaS 1*) and the allowed TW is set to 6 hours for all applications. Figure 5 presents the energy consumption (blue bars) and the percentage of energy saving (white curve) when varying the distribution of the contract selection. It also displays the sum of all applications monetary cost (orange bars) as defined in Figure 4. For this experiment, an additional execution contract (C4) is used to define the selection of the less energy consuming contract between the 3 proposed contracts.

Firstly, all applications execute with the same contract (three first bars). In contrast to a normal execution (C1), when all users select C2 or C3, the IaaS provider achieves respectively 2.16% and 5.14% of energy saving. It is important to keep in mind that at datacenter's scale, each percent of energy saved is greater than 1kWh. Moreover, the comparison is made with an IaaS manager that consolidates

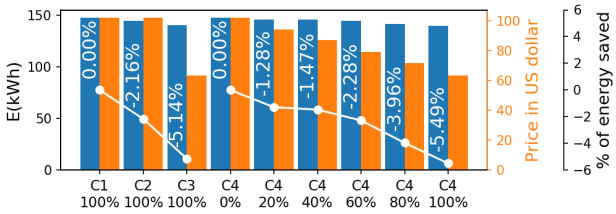

Figure 5: Energy saving percentage and total monetary cost of all PaaS application according to the distribution of the contract selection made by PaaS users.

resources and shuts down unused servers ; a scenario already more energy-efficient than currently deployed datacenter managers. In the case of 100% C2, it is not always possible to find a time slot within TW. Among the 1500 applications, 58 of them, so less than 4% had to run with a C1 contract instead of C2 (i.e. their execution were not delayed). Considering the monetary aspect, costs with C1 and C2 are the same because the resource size does not change. In the case of C3, changing the resource size reduces the total cost by almost 41%. It reveals that resources tend to be reduced in size (less expensive).

Secondly, a percentage of applications executes with the less energy consuming contract among all three (this contract is named C4 for clarity sake) and the remaining applications with contract C1. Increasing the amount of C4 contracts from 0% to 100% shows an increasing percentage of energy saving going up to 5.49%. In the 100% C4 scenario, there is 91.2% of C3 because the high flexibility of C3 shows lower energy consumption in most cases. As C1 and C2 contracts can consume less than C3, 100% C4 saves more energy than when using only C3 contracts. From a monetary point of view, executing all PaaS applications is less expensive when more of them execute with the C4 contract.

Results show that, in this context, users acceptance to delay and change the size of their applications allows to reduce the energy consumed by the IaaS provider and to lower the monetary cost for PaaS users.

2) Impact of the Time Window on Energy Consumption: In the second experiment, we show the link between the allowed time window TW and the energy savings. Figure 6 presents the results of an IaaS where all applications execute with the C2 contract and TW increases from 1 to 13 hours in 2 hours steps.

It shows that a TW of more than 3 hours is required to reach around 2% of energy savings but remains stable beyond this value. The red line shows the number of C2 contracts that failed to find a delayed time slot. In this context, a TW greater than 7 hours is necessary to have all VMs successfully delayed. This experiment shows that a minimal length of TW is required to save energy, but a too large one does not provide additional savings.

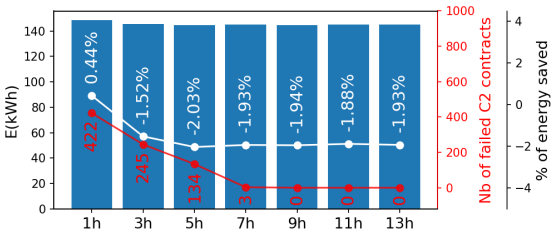

Figure 6: Percentage of energy saving with C2 contracts according to the allowed time window.

3) Increasing the Number of IaaS providers: The goal of the third experiment is to express the impact of the number of IaaS used by the PaaS on the trend on delaying applications execution. This number increases from 1 to 4 IaaS providers following the configuration given in Table II, the TW is fixed to 6 hours, and all applications execute with contract C2. In Figure 7, a cumulative distribution function (CDF) is used to plot the 1500 execution delays for each number of IaaS.

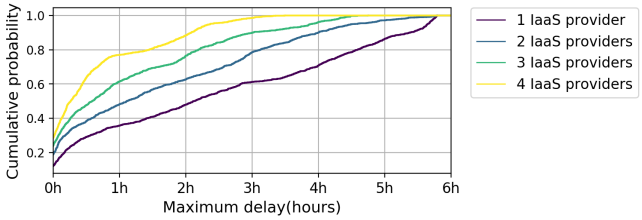

Figure 7: CDF of application delays according to the number of IaaS providers when all applications execute with contract C2.

The figure shows that whatever the number of IaaS providers, the CDF always reaches 100% before 6h of TW. It means that the system is able to keep execution delays within the time window constraint. It also shows that each time an additional IaaS provider is available, the system is able to execute more applications with a smaller delay. For example, less than 40% of applications have a maximum delay of 1h with 1 IaaS provider whereas 3 IaaS providers cover about 60% of applications for the same delay. In this scenario, 4 IaaS providers allow to have almost all applications with a delay that barely exceeds 3h (50% of the time window limit).

V. Discussions

Changing the amount of resources is possible as long as the duration table is provided. To obtain this table, users require to execute their applications on all possible VM sizes or to follow the approach presented in [7]. A more convenient solution would be to automate this process by learning from previous executions. The PaaS would record all application execution times for each VM size.. With a learning process, the PaaS would then be able to estimate applications execution time depending on resource size.

PaaS users wait until completion time to get their results. It corresponds to the sum of the delay and the execution length, both varying depending on the users flexibility. Thus, with our system, users wait for a longer time than usual but our evaluation shows two major benefits. First, it allows users to contribute to the reduction of datacenters energy consumption. And second, flexibility on the resource size tends to lower applications monetary cost.

Each execution contract comes with an energy prediction that can differ from the energy actually consumed after execution. Out of the 1500 applications, only 2.93% of them have a perfect energy prediction with our model. For about 79% of applications, the prediction is over-estimated. Indeed, while waiting for the execution to start, the server might have received other reservations. A higher vCPU reservation reduces the idle energy part attributed to the application's VM (see Eq. 2). The remaining 18% represents applications consuming more than predicted. The reason of this difference is because the continuous VM allocation on servers and the IaaS consolidation algorithm can make some servers to be better for hosting a VM than the one selected during the RA generation. Such a change can make a VM run on a server less used than predicted and thus it accounts from a larger part of the server's idle energy. The difference between predicted and consumed is not an issue because the overall prediction is over-estimated, and as the experiments show, these energy estimations provide significant savings.

The proposed system would not work in reality without a certification of the process. An audit is required in order to avoid IaaS providers from tampering their energy prediction so that they can increase their profit. At the PaaS level, it would force the filtering component to be impartial with contracts from different IaaS providers.

VI. Related Work

To our knowledge, a few studies exist on the topic of IaaS-PaaS co-design for reducing datacenter energy consumption. In their position paper [3], the authors express that an enhanced interaction between the PaaS and the IaaS layers could help achieving energy savings. Dupont et al. [4] present an extension of the traditional SLA that includes a description of the flexibility of applications deployed in cloud environments. This approach allows to have temporarily reduced or increased performance according to the availability of renewable energies. In their approach, the authors make live optimizations, such as resources scaling, while in our work we opt for temporal consolidation of resources that do not change in size over time. Unfortunately, their proposition does not present validation results. In [15], Djemame et al. propose a self-adaptive IaaS-PaaS architecture that allows to run energy-efficient cloud operations. Applications are defined with complex user-defined SLA rules that express constraints on performance, energy and price aspects. The PaaS layer negotiates and compares offers

from multiple IaaS providers and selects the best one. These solutions are similar to our proposition as they enhance the relation between IaaS and PaaS layers, and take end-users into account. However, the complex mechanisms proposed to users might prevent them from using such systems. We believe that systems including end-users should be designed to be as simple as possible to use. Additionally, the temporal placement of VMs based on resource usage prediction is not considered in these studies. In a non-virtualized datacenter context, Hsu et al. propose a solution for the waste of resources in datacenters by looking at the waste of power budget [16]. Their idea is to group jobs with asynchronous peak times under the same power node, thus allowing to reduce the peak power of each server. While their objective is similar to our waste reduction goal, we focus on resource wastage instead of power and we consider two user parameters: resource size and temporal placement.

VII. Conclusion

In this work, we propose to benefit from the flexibility in time and size of PaaS applications in order to reduce the energy consumption of underlying IaaS datacenters. Our approach enhances interactions between PaaS and IaaS and includes end-users in the resource allocation decisions. The PaaS layer negotiates execution contracts with multiple IaaS providers for executing users' applications. End-users are then proposed to select between 3 different execution contracts that allow to save more or less energy.

Through our simulation-based evaluation, we validated the gain in energy of our approach using real workload traces and real application execution logs. Allowing PaaS applications to be flexible in terms of execution delay and size allows to save up to 5.49% of energy at datacenters' level in comparison with an already energy-efficient cloud scenario where VMs are consolidated and unused servers are powered down. This energy saving is achievable by only delaying application executions up to a maximum of 6h.

As future work, we would like to enhance the inclusion of PaaS users by providing an additional flexibility in the means of dynamic scaling up and down of virtual resources. Similarly to [4] and the lever we proposed in [8], users would be able to set triggers in their application execution to add/remove resources on the fly in order to adapt their size with the current load. These predictable resource updates, motivated with a reward system, may favor improved consolidation and thus increase energy savings.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] A. S. Andrae and T. Edler, "On Global Electricity Usage of Communication Technology: Trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[2] A.-C. Orgerie, M. D. d. Assuncao, and L. Lefevre, "A Survey on Techniques for Improving the Energy Efficiency of Large-Scale Distributed Systems," *ACM Computing Surveys*, vol. 46, no. 4, p. 47, 2014.

[3] A. Carpen-Amarie, D. Dib, A.-C. Orgerie, and G. Pierre, "Towards energy-aware IaaS-PaaS co-design," in *SMART-GREENS*, 2014.

[4] C. Dupont, M. Sheikhalishahi, F. M. Facca, and S. Cretti, "Energy Efficient Data Centres Within Smart Cities: Iaas and Paas Optimizations," in *Smart City 360°*, 2016, pp. 408–415.

[5] M. A. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. Cunha, and R. Buyya, "HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges," *ACM Computing Surveys*, vol. 51, no. 1, p. 8, 2018.

[6] G. M. Wamba, Y. Li, A. C. Orgerie, N. Beldiceanu, and J. M. Menaud, "Cloud Workload Prediction and Generation Models," in *SBAC-PAD*, Oct 2017, pp. 89–96.

[7] H.-W. Li, Y.-S. Wu, Y.-Y. Chen, C.-M. Wang, and Y.-N. Huang, "Application Execution Time Prediction for Effective CPU Provisioning in Virtualization Environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3074–3088, 2017.

[8] D. Guyon, A.-C. Orgerie, and C. Morin, "Energy-Efficient User-Oriented Cloud Elasticity for Data-Driven Applications," in *IEEE GreenCom*, 2015, pp. 376–383.

[9] X. Fan, W.-D. Weber, and L. A. Barroso, "Power Provisioning for a Warehouse-sized Computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, 2007, pp. 13–23.

[10] Grid'5000. [Online]. Available: https://www.grid5000.fr

[11] G. B. Berriman *et al.*, "Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics on Demand," in *SPIE conference 5487: Astronomical Telescopes*, 2004.

[12] G. Pierre and C. Stratan, "ConPaaS: A Platform for Hosting Elastic Cloud Applications," *IEEE Internet Computing*, vol. 16, no. 5, pp. 88–92, 2012.

[13] "Parallel Workloads Archive." [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[14] "The MetaCentrum 2 log." [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/l_metacentrum2/index.html

[15] K. Djemame, R. Bosch, R. Kavanagh, P. Alvarez, J. Ejarque, J. Guitart, and L. Blasi, "PaaS-IaaS Inter-Layer Adaptation in an Energy-Aware Cloud Environment," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 127–139, 2017.

[16] C.-H. Hsu, Q. Deng, J. Mars, and L. Tang, "SmoothOperator: Reducing Power Fragmentation and Improving Power Utilization in Large-scale Datacenters," in *International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 535–548.